Automatic Abstraction and Fault Tolerance in Cortical Microachitectures

Atif Hashmi
University of Wisconsin,
Madison, WI, USA.
ahashmi@wisc.edu

Hugues Berry
INRIA Rhone-Alpes,
Villeurbanne, France.
hugues.berry@inria.fr

Olivier Temam
INRIA Saclay,
Orsay, France.
olivier.temam@inria.fr

Mikko Lipasti
University of Wisconsin,
Madison, WI, USA.
mikko@engr.wisc.edu

ABSTRACT

Recent advances in the neuroscientific understanding of the brain are bringing about a tantalizing opportunity for building synthetic machines that perform computation in ways that differ radically from traditional Von Neumann machines. These brain-like architectures, which are premised on our understanding of how the human neocortex computes, are highly fault-tolerant, averaging results over large numbers of potentially faulty components, yet manage to solve very difficult problems more reliably than traditional algorithms. A key principle of operation for these architectures is that of automatic abstraction: independent features are extracted from highly disordered inputs and are used to create abstract invariant representations of the external entities. This feature extraction is applied hierarchically, leading to increasing levels of abstraction at higher levels in the hierarchy.

This paper describes and evaluates a biologically plausible computational model for this process, and highlights the inherent fault tolerance of the biologically-inspired algorithm. We introduce a stuck-at fault model for such cortical networks, and describe how this model maps to hardware faults that can occur on commodity GPGPU cores used to realize the model in software. We show experimentally that the model software implementation can intrinsically preserve its functionality in the presence of faulty hardware, without requiring any reprogramming or recompilation. This model is a first step towards developing a comprehensive and biologically plausible understanding of the computational algorithms and microarchitecture of computing systems that mimic the human cortex, and to applying them to the robust implementation of tasks on future computing systems built of faulty components.

Categories and Subject Descriptors

B.8.1 [**Hardware**]: Performance and Reliability; C.1.3 [**Processor Architectures**]: Other Architecture Styles

General Terms

Algorithm, Design

Keywords

Automatic Abstraction, Fault Tolerance

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISCA'11, June 4–8, 2011, San Jose, California, USA.
Copyright 2011 ACM 978-1-4503-0472-6/11/06 ...$10.00.

1. INTRODUCTION

The original Von Neumann model of a computing unit has been a relatively nice fit for the technology evolutions of the past four decades. However, it is hard not to notice that this model is under growing pressure. The power dissipation bottleneck has made architects shift their focus to multi-core architectures but the programming bottleneck of multi-cores raises doubts on the ability to truly take advantage of many-core systems. More recently, the reliability bottleneck brings a whole new set of challenges to the table. Architects have attempted to meet all these challenges, but the proposed solutions progressively erode performance scalability. With such limitations, it now makes sense to investigate alternative models better suited to cope with this technology evolution.

Either upcoming ultra-CMOS [34] technology, or alternative technologies like nanotubes [7], share some common properties. First, the number of individual transistors/elements will continue to increase. Second, these elements will not necessarily be much faster (in some cases slower). Third, they will come with a growing number of defects and faults. Now, when one considers these properties, it is hard not to observe that nature has found a way to harness a huge number of elements with similar properties to realize complex information processing tasks.

The fact that biologists have made tremendous progress in understanding the working of parts of the brain [20] is not yet well-known to computer architects. Considering the abilities of the brain, it is clear that computer architects should leverage this information, if only for special purpose computing systems. Computer architects are uniquely positioned for this task because, unlike biologists, their goal is to steer this research towards useful computing systems and applications. Although both the elementary components and the resulting biologically-inspired computing systems are quite different from existing systems, similar approaches can and should be used to architect them. These approaches include understanding how to combine and control elementary components hierarchically into increasingly complex building blocks, defining a programming approach for these computing systems, understanding their potential applications scope, understanding the appropriate modelling level to integrate billions of components without being overwhelmed by complexity nor missing key properties, and so on.

Using the example of vision processing, and employing specific advances recorded in the neuroscientific literature, we show that it is now possible to *rebuild/replicate* certain cortical sensory tasks out of elementary neurons, thereby providing a detailed explanation of how such functions can emerge, and operate, within the brain. Next, we highlight that the corresponding architectures are based on a small set of structural and operating rules, which are local by nature, and that repetitively and hierarchically applying these

rules results in architectures capable of increasingly complex tasks. Consequently, these architectures have intrinsic *scalability* properties: they easily translate additional components into increasingly powerful processing tasks; unlike in traditional computers, scalability here means *more complex functions* rather than *faster execution*, though both can be related. These architectures also have *programmability* assets: they only rely on the repeated exposure to data, without resorting to the supervised training common in artificial neural networks. Finally, we describe a specific GPGPU-based software implementation of a hierarchical cortical network, modeled after the visual cortex, that is capable of complex visual recognition tasks in a biologically-plausible fashion [13].

Within this context, this paper makes the following novel contributions:

- We describe how the random structure and permanent learning properties of the cortical network model are intrinsically robust to permanent defects, far beyond the capabilities of traditional computing systems.
- We introduce a stuck-at fault model for cortical networks and show how it maps to permanent faults that can occur in current-generation GPGPus that are emulating a cortical network in software.
- We show how to emulate these stuck-at faults efficiently, using at-speed GPGPU execution, and validate that approach against a detailed cycle-accurate simulator (GPGPUSim).
- We demonstrate experimentally that the performance of the cortical network (measured as its recognition rate) degrades gracefully with increasingly faulty hardware, and that the network is able to transparently recover lost functionality without any reprogramming or recompilation.

2. CORTICAL STRUCTURE, OPERATIONS AND MODELING

2.1 Cortical Organization

The *neocortex* is the part of the brain that is unique to mammals and is highly developed for humans; it accounts for about 77% of the human brain (in volume) [38]. The apparent uniformity of the neocortex suggests that even though different regions specialize in different tasks, they employ the same underlying algorithm.

Neuron and Synapse. Neurons and synapses are the most well-known elementary building blocks of the brain. A neuron performs two types of operations: it *sums* its inputs and it *triggers* an output if the sum is beyond a certain *threshold*. Typical firing interval for a neuron is 20ms to 200ms [21], orders of magnitude slower than CMOS transistors switching times. A synapse is the connection between two neurons; each neuron has hundreds to thousands of synaptic connections.

Structure. Neurons are spatially organized in two ways, see Figure 1. Horizontally, they are grouped in 6 layers [15], and the convoluted form of the cortex simply corresponds to the folding of these 6 layers within a restricted volume. Vertically, they are grouped into structures known as cortical columns or hypercolumns (HCs). The HCs are decomposed into minicolumns (MCs) containing approximately 200 neurons; groups of 50 to 100 such MCs correspond to a HC. The term *cortical column* is sometimes used for both types of columns, though, in biology, it usually refers to HCs.

Within and across cortical columns, there are numerous forward, backward (called feedback) and lateral connections. These connections can be either excitatory (they can increase the output of the target neuron) or inhibitory (they decrease its output). This multi-directional flow of information can be observed at different levels of granularity: within a cortical column, across cortical columns, and across regions which derive from the hierarchical organization of cortical columns.

2.2 Cortical Operations: Automatic Abstraction

While the cortical structure of certain regions of the brain, such as the visual cortex, has been investigated for a long time, quantitative models, consistent with physiological data, and capable of accounting for complex visual tasks, were proposed only recently [32, 33]. Such models are particularly valuable because they provide an implementable explanation for *automatic abstraction*. It is believed that cortical regions operate by progressively abstracting and manipulating increasingly complex notions throughout the neural hierarchy [26].

For instance, from the set of pixels of an image, the visual cortex will first identify segments, then elementary shapes such as angles and intersections, and increasingly complex combinations, such as objects found in our environment, see Figure 2. This automatic abstraction capability for various inputs (e.g. visual, auditory, and olfactory) partly explains why the neocortex still outperforms traditional computers for a number of tasks. Emulating such capability is thus a major step in building computing systems that can compete with at least some processing characteristics of the brain.

2.3 Biological vs. Artificial Neural Networks

In terms of developing neural models, the machine-learning community pursues a different goal than the neurobiology community: to build the most efficient classification algorithms. Instead, the neurobiology community seeks to build models of neural networks which shall ultimately emulate the whole range of biological neural networks' capabilities, classification being only one of them. Our end goals are more in sync with the neurobiology community: emulate biological neural networks with the goal of reproducing a large range of their capabilities, along with the additional goal of achieving computational efficiency.

There are additional significant differences between biological and artificial neural networks. Both types of networks typically rely on different *learning* strategies. ANNs often rely on back-propagation for learning classification tasks: the correct answer is known and is fed back through the network by adjusting the synapse weights based on the network error; this form of learning is called *supervised learning* [14]. Biological neural networks rely on what the machine-learning community calls *unsupervised* learning: the correct answer is not known, but the network learns through repeated exposure to the input via local learning rules, i.e., Hebbian learning [4]. There may also exist indirect feedback, e.g. reward/punishment (*reinforcement* learning) [37].

Another major difference between biological and artificial neural networks is the attention given to the *network structure*. ANNs often rely on full connectivity, or random structures [14]. Recent progress in neurobiology [10, 3] show that the network structure and the nature and arrangement of connections are complex and play a major role in the abstraction capabilities of the network.

Finally, both single and multi-layer perceptron models are highly intolerable to permanent hardware defects. Emmerson et al. [8] show that even a slight amount of faulty behavior (either in the perceptron or the connections) can significantly deteriorate the recognition rate of these models.

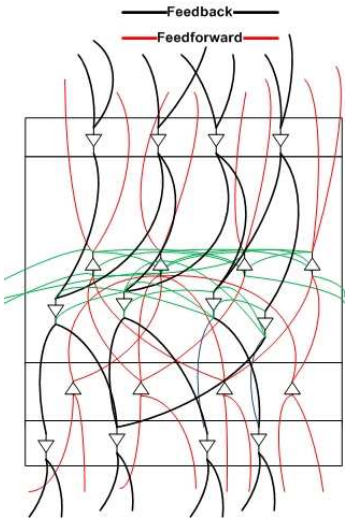

Figure 1: *Forward, feedback and lateral connections between neurons and cortical columns.*

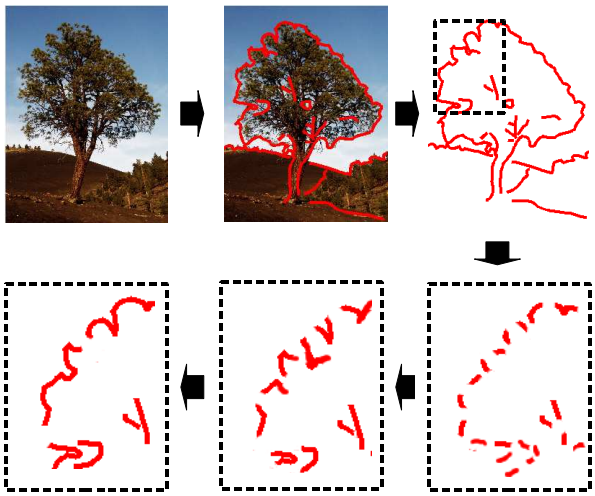

Figure 2: *Increasingly complex visual abstractions (segments, angles and long segments, complex shapes, etc.).*

2.4 Cortical vs. Traditional Microarchitectures

The externally-observable operation of many systems can be reproduced with a behavioral model that operates at an appropriate level. As architects, we are familiar with many examples of such behavioral models. Increasingly high-level models often aggregate the behavior of lower-level elements in time and/or in space. For example, a gate-level model of a digital circuit removes details related to individual transistors and does not model transient switching behavior, reporting only steady-state logic values. Typically, the precision of these models varies inversely with their computational demands, and an appropriate model must be chosen to satisfy both reasonable time to completion and sufficient precision.

Sandberg and Bostrom provide a thorough introduction to neural modeling, and identify eleven different levels of model, ranging from ANNs all the way to molecular dynamics and even quantum-level simulation [30]. They argue that brains can be emulated without higher-level algorithmic understanding: as long as biological details are measured carefully and replicated faithfully, a feline brain, or even a human brain, will "boot up" and work as expected. A recent experiment showed that a large-scale supercomputer (IBM Blue Gene/L) possesses the computational throughput for modeling a rat cortex using this approach within an order of magnitude of real time (9x slowdown) [1]. Also, the DARPA Synapse [6] program has set a goal of scaling emulation up to a feline cortex.

In contrast, as architects, we want to build cortically-inspired computing systems which emulate selected functional subsets of the human cortex. This requires detailed high-level algorithmic understanding of the cortical properties, rather than simply precisely reconstructing the biological baseline. It also requires high computational efficiency, which suggests developing high-level behavior models emulating cortical functionality. The pitfall is development of high-level but unfaithful models which fail to emulate the target functions because they do not capture the key aspects of their biological implementation. For that reason, such high-level models must be validated against lower-level, less computationally efficient, but biologically plausible models.

This paper is a first step in that direction; we describe a cortical column-level model that matches existing physiological data for structure and operations, and that is capable of one critical aspect of cortical computation: feed-forward sensory processing and automatic abstraction along the lines of the visual cortex (which is relatively well understood in the neuroscience literature). We then provide and evaluate our implementation of the cortical model and show that the online automatic abstractions and robustness properties of biological networks translate into an intrinsic tolerance of the model to permanent hardware defects on a GPGPU architecture.

2.5 Silicon-Based Implementation

We do not present a specific silicon-based implementation in this article, because the recent progress in neurobiology, which motivates this work, relates more to the structure and connections between neurons, than to the behavior of neurons and synapses themselves. As a result, the neural arrangements presented in this article can be readily implemented in silicon by leveraging the large body of work on hardware implementation of artificial neural networks [16, 31]. Recent hardware implementations of ANNs, especially those which leverage analog or hybrid digital-analog structures [31], provide very low power consumption. Finally, one of the major assets of ANNs is their inherent robustness to transient faults. They are robust to transient faults because information/decisions are averaged out over multiple neurons, so one neuron temporarily providing erroneous information has no catastrophic impact [9].

3. A BIOLOGICALLY PLAUSIBLE AND EFFICIENT MODEL FOR A CORTICAL MICROARCHITECTURE

In this section, we present our cortical column model which builds upon the cortically inspired hierarchical learning model proposed by Hashmi et al. [11, 12, 13]. As mentioned in Section 2, the key feature of our model is to implement the notion of *automatic abstraction*. This notion was first proposed by neuroscientists in the HMAX model, see Riesenhuber et al. [27]; this model itself relies on physiological data of large-scale biological neural networks. However, in that model, the synaptic connections and values are precisely preset. As a result, this model does not have the robustness capabilities that we are seeking (losing a neuron or

a synapse can drastically affect functionality), nor can such precise wiring occur in the biological case either. Therefore, if we can implement a model with similar abstraction capabilities by only relying on stochastic connections, we achieve the desired robustness capabilities; and in the process, we actually improve the biological resemblance of the model.

3.1 Input and Receptive Field

In the mammalian brain, a nerve path transfers visual data from the retina to the LGN (Lateral Geniculate Nucleus) cells [20]. LGN cells detect *contrast*: they react strongly to an illuminated point surrounded by darkness (on-off cells) or conversely to a dark point surrounded by light (off-on cells). These cells are spatially distributed with on-off and off-on LGN cells intertwined [28], roughly operating like a pixel sensor. The output of the LGN cells is then fed to the rest of the visual cortical hierarchy.

Modeling biology, before exposing the visual inputs to our model, they are preprocessed using the LGN transform. In our model, we consider a regular spatial distribution (grid-like) of LGN cells, i.e. one on-off and one off-on cell per pixel. We have also experimented with more random distributions of LGN cells without noticeable differences. From these experiments, we have concluded that the only important factor is the spatial density of LGN cells with respect to the image resolution.

The *receptive field* is a tool used by biologists to describe what a cell (LGN or any neuron in the visual cortex) "sees". The receptive field of an LGN cells is two concentric circles made up of retinal cells. An on-off LGN cell shows high activation if the retinal cells in the middle of its receptive field have high activation while the surrounding retinal cells have low activation, i.e. light surrounded by darkness. On the other hand, an off-on LGN cell shows high activation if the retinal cells in the middle of its receptive field have low activation while the surrounding retinal cells have high activation. The receptive field of LGN cells is shown on the bottom of Figure 3.

3.2 Connectivity

As explained before, neurons are arranged into layers within a cortical column (or minicolumn). Furthermore, neurons connect to neighboring neurons (either within or between columns) with a Gaussian probabilistic law, i.e., the probability of connection decreases with distance. Empirically, the connectivity is actually fairly dense (about 70%), almost uniform, within a given neighborhood [19], and drops afterward [28]. This suggests that neurons within a restricted neighborhood are almost fully connected.

Based on these biological findings, in our model, we group the minicolumns (MCs) into hypercolumns (HCs) with the following rules (see Figure 4): First, a MC within a HC is fully connected to all the MCs within the same HC via lateral inhibitory connections. This emulates a winner-take-all sort of a behavior among the MCs within an HC. Second, all the MCs within a HC share the same receptive field, i.e. the MCs belonging to the same HC are connected via excitatory feedforward connections to the same MCs of the lower level HCs (albeit with different synaptic weights). Figure 4 demonstrates the connectivity among the MCs within a HC and also the connectivity among HCs at various hierarchical levels. It should be noted that the MCs in the upper levels receive inputs from multiple lower level MCs. Moreover, the number of HCs decreases rapidly from one level to the other, corresponding to a hierarchy where each level width quickly shrinks. This organization of the topology is a direct extrapolation of the hypothesis proposed by Hubel and Wiesel for the connectivity of the first levels [17].

3.3 Evaluation of Minicolumn Activity

Activity of a MC depends on the activations of its input MCs weighted by their synaptic weights. Formally,

$$x_i(t) = f\left(\tilde{g}_i(t)\left(\sum_j C_{ij}(t) - T\right)\right)$$

$$C_{ij}(t) = \begin{cases} \text{-2.0} & \text{if the weight } \tilde{W}_{ij}(t) \text{ is low} \\ x_j(t)\tilde{W}_{ij}(t) & \text{if the weight } \tilde{W}_{ij}(t) \text{ is high} \end{cases}$$

The output $x_i(t)$ of MC_i is between 0 and 1 (inclusive) and we define MC_i as active at time t if its activity $x_i(t) > 0.7$. $f(X) = \frac{1}{1+e^{-X}}$ is a non-linear activation function also used by traditional ANN models. The normalized weights and gain are defined as $\tilde{W}_{ij}(t) = W_{ij}(t)/\Omega_i(t)$ and $\tilde{g}_i(t) = 10 \times \Omega_i(t)$. Here, $W_{ij}(t)$ is the weight of synapse $j \rightarrow i$ at time t. $\Omega_i(t) = \sum_j W_{ij}(t)$ is the sum of all the synaptic weights of a MC. Note that weight normalization $\tilde{W}_{ij}(t)$ is a (simple) emulation of the "synaptic scaling" phenomenon observed in several regions of the brain [18] and that of the gain was inspired by the so-called "intrinsic plasticity" (i.e. activity dependent modification of the neuron gain) observed in the cortex [25]. T defines the robustness of a MC to noisy input and can take a value between 0 and 1. For our experiments, we set it to 0.95.

Simple ANN models usually define the input of the activation function simply as $\sum x_j W_{ij}$. The expression of C_{ij} can be seen as a reflection of the non-linear summation properties observed in some dendrites [23]: when the weight W_{ij} is low, the contribution is very negative (non-linearity). We empirically observed this non-linearity to be necessary for proper functional behavior.

With respect to the notion of "low" and "high" weights in $C_{ij}(t)$, it must be noted that there is a form of permanent competition between the synapses of a neuron, i.e., a form of zero-sum game which is akin to permanent normalization. Such rescaling only occurs for large synaptic weights. The definition of "low" and "high" weights are thus:

$$\begin{array}{l} \text{Low Weight if } \tilde{W}_{ij}(t) < 1/\Omega_i(t) \\ \text{High Weight if } \tilde{W}_{ij}(t) \geq 1/\Omega_i(t) \end{array}.$$

3.4 Evaluation of Hypercolumn Activity

In our model, a HC is an abstract representation of a group of MCs that are tightly bound together via lateral inhibitory connections. A HC gets active if any of the MC within that HC gets activated. The MCs within a HC follow a winner-take-all approach i.e. if multiple MCs within a HC get active, the MC with the strongest activation inhibits the rest of the activated MCs. Thus at any moment in time only one MC within a HC is active. Furthermore, the inhibited MCs modify their synaptic weights to decrease their correlation with the present input so that they do not get activated by the same input pattern in the future. This results in MCs within a HC learning independent/unique inputs exciting the receptive field of the HC..

3.5 Learning Through Repeated Exposure and Random Firing

As mentioned in Section 2, Hebbian learning [4] is a dominant form of learning in large-scale biological neural networks. With Hebbian learning, if an input of a neuron is strongly activated, and that neuron itself has a strong output, then the synapse (synaptic weight) corresponding to that input is reinforced. Intuitively, if the input is strong at the same time as the output, it means that input plays a significant role in the output and should be reinforced. According to this definition, the synaptic weights of an active MC

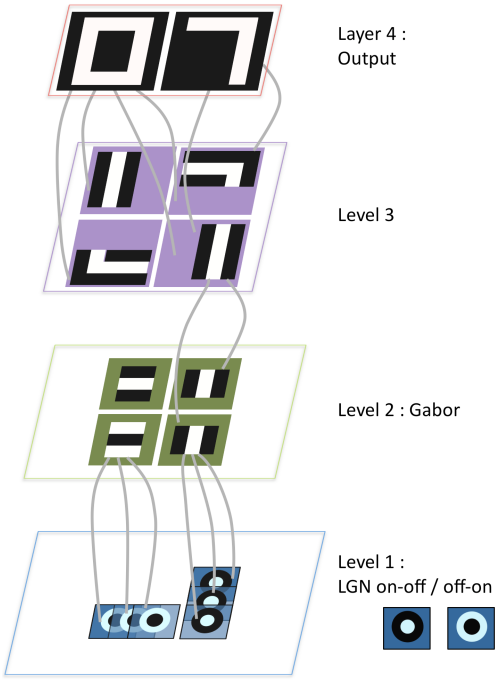

Figure 3: *Illustration of automatic abstraction of complex objects applied to visual processing with a 4-layer hierarchy. For clarity, the MC-HC structure is not illustrated and only a fraction of the connections is illustrated.*

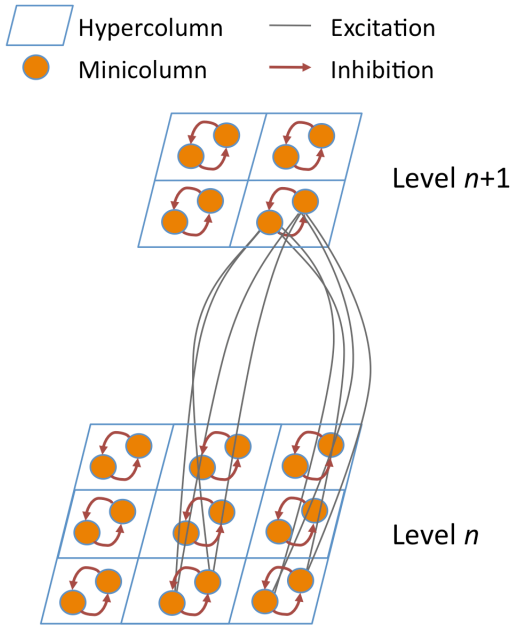

Figure 4: *Organization of the connectivity. For clarity, this scheme only considers two MC per HC and only shows some of the connections of a single HC in level $n+1$.*

corresponding to its active inputs are increased (emulating long-term potentiation), or decreased if the inputs are inactive (emulating long-term depression). This weight modification is only applied to active minicolumns x_i in accordance with Hebbian learning.

As a result of Hebbian learning, at each level, minicolumns will progressively react most strongly to inputs they receive repeatedly, in effect *learning* them. In the visual cortex, these inputs correspond to shapes, which get increasingly complex in the upper levels. The shape "memorized" by a neuron can actually be "displayed" using a receptive field, see Figure 3.

The capability of the network to distinguish between a large number of shapes, and to achieve rotation, translation and scaling invariance, rests in its capacity to learn a large number of different shapes. Since all MCs within a HC receive the same input (as they share the same receptive field), the main distinction among MCs within a HC rest in their connectivity with the MCs in the lower levels. This connectivity is modeled by modulating the strengths of synaptic weights corresponding to various inputs to a MC. Strong synaptic weight corresponding to a lower level MC represents strong connectivity while 0 weight synapses are equivalent to no connectivity.

Unlike traditional Hebbian learning models, we do not rely on random initialization of synaptic weights for initial network connectivity. One of the main disadvantage of this approach is that sometimes the resulting networks might not be capable of learning some of the input patterns as their initial connectivity does not allow it. Rather, in our model, the synaptic weights of all of the MCs are initialized with very weak random values. This suggest that the MCs in our model do not assume any initial connectivity and the whole network is like a blank slate. We leverage on the random activation property of the MCs for inducing random initial conditions, and for perturbing the inputs of the MCs as the learning process occurs. More precisely, at each time step, each MC has a small probability to become active even if its input do not justify it. This corresponds to allowing random firing of the MC. If a MC randomly fires, its synaptic weights corresponding to its active inputs are reinforced. Thus, through this random firing behavior, initial network connectivity is established. We stop random firing for MCs with strongly established synaptic weights for feed-forward communication. We empirically observed that this random firing allows a great variety of shapes to emerge, but that stopping random firing is necessary to stabilize columns which have converged.

The biological origins of random firing and the fact it stops after repeated activity are the following ones. Neurons receive synaptic inputs from all types of connections: forward, lateral, feedback. As long as the forward synapses are weak, the combination of these inputs creates a "synaptic noise", akin to random firing. When the forward connections become strong, because the neuron has "learned" a feature, they become dominant and the neuron output is no longer affected by the remaining "synaptic noise". As a result, it appears as if random firing has stopped, though it still exists but no longer has a significant impact in the network's behavior.

3.6 Automatic Abstraction

We now explain how the notion of automatic abstraction is implemented in HC hierarchy, using our visual cortex example. Each MC of the first level "samples" the input pixels within its receptive field, see level 1 in Figure 3. The subsequent MCs progressively respond to the activations of the MCs within their receptive fields and thus learn complex shapes.

Emergence of shapes. The first-level MCs connect to the output of the LGN cells. Based on the initial random activation of the MCs, connections between the first level MCs and the LGN cells within their receptive field are established. As explained in Section 2.2, the LGN cells detect contrasts, and especially extract

contours. These contours of most natural shapes decompose into tiny segments at a fine enough granularity. The first-level MCs, initially by virtue of random firing respond to such segments, and afterwards are strengthened through Hebbian learning. These segments progressively emerge as some of the dominant shapes in the first-level columns, (see Figure 3). This behavior is supported by biological evidence of the existence of Gabor filters in the first layers of the visual cortex [17]. The subsequent levels continue the same process and learn increasingly complex shapes. For instance, combinations of segments can produce crosses, angles, and other complex shapes (see Figure 3).

Filtering. The MCs higher up in the hierarchy correspond to the aggregate information (sum) of an increasingly high number of lower level MCs. As a result, their receptive field rapidly becomes a murky combination of more simple shapes, and does not carry a crisp semantic. Here, lateral inhibitory connections enable a necessary filtering role by silencing weak minicolumns, and allow crisper shapes (richer semantic) to emerge in upper-level columns. These lateral inhibitory connections implement a form of *max* operators among clusters of MCs. Formally, this lateral inhibition is again implemented using a Hebbian learning rule i.e. if MC_i is active ($x_i(t) > 0.7$) but there is at least one MC (say MC_K) in the HC such that $x_K(t) > x_i(t)$, then MC_i undergoes lateral inhibition from K, which means in our model : (1) $x_i(t) = 0$ and (2) the weights incoming to MC_i from active synaptic units are downscaled. This form of modification implements competition within the HCs and corresponds to so-called presynaptic lateral inhibition [36].

3.7 Summary

In summary, with such a model, there is no need for *a-priori* specification or knowledge of the information to be extracted from input data. The information is progressively abstracted into increasingly complex notions. Which complex notions emerge will depend on the network structure, the initial conditions and the characteristics of the input data. In other words, this process implements a form of "learning by example".

4. MODEL EVALUATION ON GPGPU

4.1 Model Implementation

While the model described in the previous sections may eventually be realized with specialized hardware, we have investigated fitting a software version of that model on a currently available architecture. The most attractive architecture we have encountered so far is the general purpose graphics processing unit (GPGPU), specifically NVIDIA's CUDA. In this programming model, highly parallel workloads can be processed on hundreds to thousands of CUDA threads. In current top-end CUDA devices, groups of threads are scheduled to run on a streaming multiprocessor (SM) which is composed of eight in-order cores and 16KB of fast-access shared memory [5]. CUDA makes it easy for programmers to optimize their applications through a number of different methods, including memory access coalescing and using the shared memory space as a fast-access user-managed cache [29].

Nere et al. [24] have successfully demonstrated that the cortical model of Hashmi et al. [13] maps well onto the CUDA architecture. We use this CUDA based implementation for our experiments. By mapping a single minicolumn to a CUDA thread, we can have thousands of minicolumns concurrently active on a GPGPU. Since the minicolumn's firing is based on the dot-product evaluation of the input and minicolumn weights, the model is an example of a high-throughput data-intensive application CUDA was invented for. Finally, the shared memory space per SM is ideal for fast lateral communication between neighboring minicolumns.

Section 3.6 describes the cortical architecture as having different hierarchically organized components, composed of minicolumns and hypercolumns. Similarly, NVIDIA's CUDA framework consists of a hierarchical organization, with threads, cooperative thread arrays (CTAs), and kernel launches. The GPU-accelerated code translates the components of the cortical architecture to the CUDA framework. With such an organization on CUDA, the minicolumns in a hypercolumn can easily synchronize as well as laterally communicate and share receptive field inputs in the fast access shared memory space, as seen in Figure 5.

Initially, we evaluate the recognition accuracy of our model by creating a hierarchy of multiple hypercolums and exposing it to a subset of handwritten digit images obtained from the MNIST handwritten digit database [22]. Our training data consists of 100 variations of each of the digits (a total of 1000 images). For this experiment, we create a hierarchical hypercolumn network with 5 levels. This hierarchical network contains 24 hypercolumns each with 15 minicolumns at level 0, 12 hypercolumns each with 20 minicolumns at level 1, 6 hypercolumns each with 20 minicolumns at level 2, 3 hypercolumns each with 15 minicolumns at level 3, and 1 hypercolumn with 15 minicolumns at level 4. After 15,000 training epochs, the hierarchical network exhibited recognition of each of the handwritten digit images in the training dataset with 100% accuracy. This means that, after 15,000 training epochs, each of the handwritten examples in the training dataset (see examples in Figure 6) is correctly assigned to the digit (0-9) it actually represents. The software model implemented on a GeForce 9800 GT GPU achieves a recognition rate of 40 characters per second, which is comparable to the biological example [35].

The test set of MNIST database consists of 10,000 images of handwritten digits. For the test set, our model achieves an average recognition rate of 85% when trained with 1,000 digit images. We note that this is not comparable to the present state-of-the-art ANN handwritten digit recognition applications. The main reason is that, various parameters of these ANN applications are carefully tuned in order for these applications to achieve less than 1.0% error rates. Thus, the performance of these applications is quite specific to the training datasets. On the other hand, our model does not rely on any carefully tuned parameters which makes it more general and suitable for a wide variety of training datasets. This means that for our model, the same network can be utilized for robust recognition of digits, synthetic images, and real life images. It should also be noted that the purpose of this study is not be beat the recognition accuracy of the state-of-the-art ANN handwritten digit applications. Rather, we intend to show that the biologically plausible connectivity and learning rules make our model far more tolerant to permanent hardware defects as compared to traditional ANN applications.

4.2 Robustness and Inherent Fault Tolerance

Permanent defects and transient faults are not only a concern for future architectures, but are already a prevalent issue in some of the latest systems. For instance, the NVIDIA Fermi is the first GPU architecture to provide SECDED error correcting code for all DRAMs, caches and registers. Permanent defects, at design time or during the chip lifetime, are also expected to further increase in the future.

Currently, applications programmed for GPU chips like Fermi or Tesla, assume that all cores of the GPU function correctly. If any of the 512 shaders (cores) of a Fermi chip becomes dysfunctional (increasingly likely as the number of cores increases), it would be

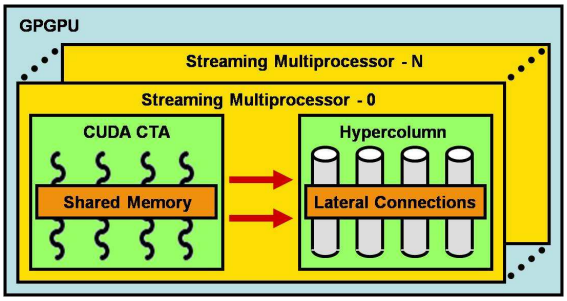

Figure 5: *Mapping a hypercolumn to a CUDA CTA.*

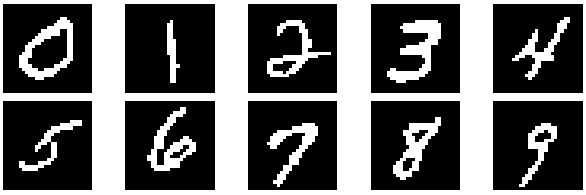

Figure 6: *Sample of handwritten digits obtained from MNIST database.*

necessary to rewrite applications so that no task is mapped to faulty shaders, or the compiler would have to perform that remapping automatically. Here, we consider that the programmer or compiler is given explicit control over the shader mapping, though no such ability is yet present in current generation GPGPUs. For instance, all existing vision recognition applications written for GPUs would have to go through that reprogramming or recompilation process. Even vision recognition applications based on artificial neural networks would suffer from the same limitation. While ANNs are conceptually neural networks, neither their back-propagation learning process nor their software implementation as imperative array-based computations are defect tolerant.

The software implementation of our model for the GeForce 9800 GT GPU preserves the key concepts of the model: the connections (synaptic weights) are initialized with weak random values, operators (sum, max) are implemented in a robust manner through a set of synapses, there is no central control nor supervision for the learning process since it happens in a distributed manner, yet it can implement complex tasks such as vision recognition. Thanks to these properties, the software implementation of our model is inherently robust to faulty GPU hardware. It can function properly and thus, unlike most other applications, it can take advantage of the GPU, *without requiring any reprogramming or recompilation*, even if one or several of the cores is dysfunctional and has to be deactivated. All that is needed is to periodically retrain the application so that it adapts to the new configuration of the faulty hardware, but again without specifying that configuration; the learning process will automatically adjust to the faulty hardware. Naturally, the present article only makes a demonstration of such robustness for vision recognition, but part of our motivation for a generic cortical model is to apply it to the broad range of tasks that large-scale biological neural networks are known to tackle.

4.3 Fault Identification and Detection Model

In the case of a biological network (the neocortex), the fault model is quite simple: if neurons or minicolumns become defective, they stop generating any activations and eventually die out. When this type of fault occurs, other neurons/ minicolumns modify their synaptic weights to detect and interpret the feature that was previously recognized by the damaged neuron/ minicolumn.

In our software implementation, each MC runs on a shader core, which can have multiple failure modes. For this work, we set aside catastrophic failures (e.g. short circuits, which crash the entire GPU) and intermittent faults, which may lead to transient degradation in recognition rate, but are handled gracefully by the forgiving nature of the cortical column algorithm (since our model uses repeated exposures of the same image for learning and recognition, intermittent faults and transient errors are averaged out). Instead, we focus on permanent faults that corrupt the computation of the minicolumn. Ultimately, due to the sigmoid nature of the output activation functions these faults will manifest themselves as the MC either not firing when it should (a stuck-at-zero fault), or firing when it should not (a stuck-at-one fault).

A MC stuck-at-zero behaves the same as a damaged neuron/ minicolumn in a biological network: since it is not generating any activity, its functionality is automatically taken over by the neighboring MCs. On the other hand, the MCs that are stuck-at-one can severely degrade the performance of our network, since all the information being generated by the MCs hierarchically below the defective MC is masked by the fault. Thus, the stuck-at-one condition must be detected and the MC deactivated.

To detect stuck-at-one conditions, we recompute the response of the winning minicolumn on two neighboring shaders, and use voting to determine which, if any, of the shaders is defective. To minimize overhead, we perform this recomputation only intermittenly, once every 100 iterations. This is a reasonable optimization, since permanent faults are rare and detection latency is not critically important for our image recognition application. In other applications, a higher-overhead, lower-latency detection mechanism may be warranted. A shader is disabled if two of its neighbors disagree with its result. In subsequent iterations of the model, all neighboring MCs (in the same hypercolumn) as well as upstream MCs in the next level of the hierarchy ignore the output of the defective MC.

At this point, our model utilizes the idea of automatic abstraction and random firing to relearn the features being recognized by the MCs running on the defected shader core. It should be noted that since all the MCs within a HC share the same receptive field, MCs connected to defected MCs need not be reconnected as their output is already being exposed to multiple MCs at the upper level in the hierarchy.

4.4 Fast Emulation of a Faulty GPU using a Fault-free GPU

Here, we demonstrate that we can emulate the behavior of a faulty GPU by simply deactivating one or more MCs within our model and running the model on a fault-free GPU. Demonstrating this equivalence allows us to perform long-running and large-scale experiments on the robustness of the model without incurring the overhead of detailed simulation of a faulty GPGPU.

In order to show this equivalence, we use the simulator GPGPUSim configured to emulate the real GPU we validate against and later use, the aforementioned GeForce 9800 GT; this GPU has 14 multiprocessors with 8 shader cores in each one. Initially, we use GPGPUSim [2] to simulate the effects of deactivated shader cores (scalar processors) on our hypercolumn model. For our experiments, we add a feature in GPGPUSim that allows us to deactivate a given shader core. The output of deactivated shaders remain unchanged at zero.

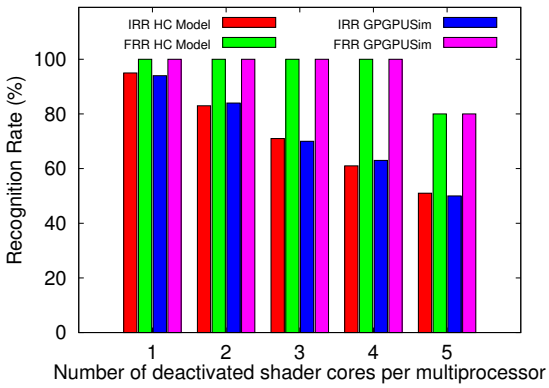

Figure 7: *Effects of shader core deactivation on hypercolumn recognition rate.*

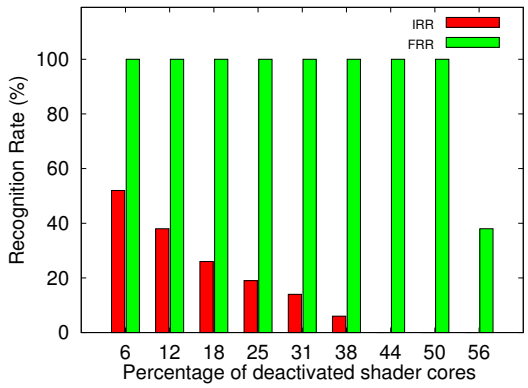

Figure 8: *Effects of shader core deactivation on the hierarchical hypercolumn network recognition rate.*

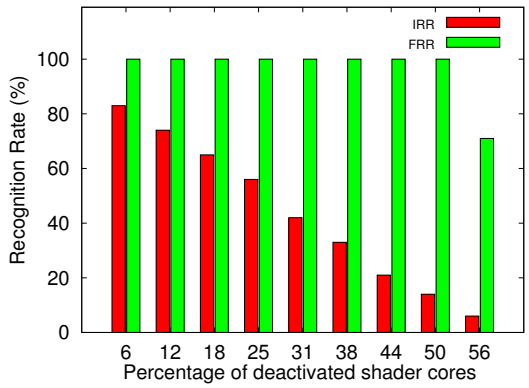

Figure 9: *Effects of shader core deactivation on hypercolumn recognition rate with redundant hierarchies.*

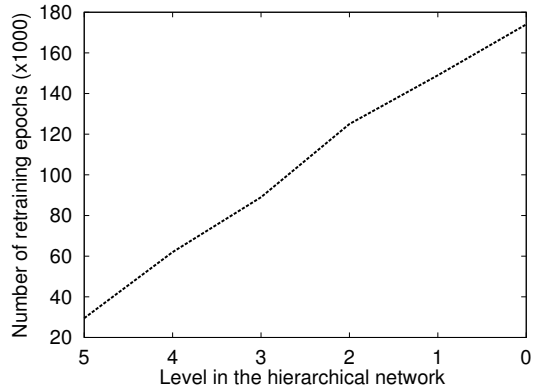

Figure 10: *Number of retraining epochs required to achieve FRR with spatially localized shader core deactivation.*

On GPGPUSim, we perform the following experiment. We initialize a single hypercolumn with 32 minicolumns with a receptive field of size 3×3. We expose this hypercolumn to 15 unique patterns of size 3×3. After a few training epochs, the hypercolumn recognizes each of the unique patterns. This means that, out of 32 minicolumns, 15 adjusted their weights so that they would fire for one of the unique features in the training set.

At this point, we inject permanent faults in randomly selected shader cores. These shader cores are then detected and deactivated. Then, we evaluate the impact of deactivating the defected shader cores on the recognition rate of the hypercolumn. Note that in our model implementation, each hypercolumn maps onto a multiprocessor and each minicolumn maps onto a shader core. Thus, 4 minicolumn threads map onto a single physical shader core, so that deactivating a shader disables 4 minicolumns. The results of this experiment are reported in Figure 7, labeled as `GPGPUSim`. All the results are averaged over 20 trials unless otherwise stated. The initial recognition rate (IRR) is the recognition rate immediately after the shader cores are deactivated, before any retraining, while the final recognition rate (FRR) is the recognition rate after retraining. We then perform the following experiment on the real GPU, labeled as `HC Model`. We deactivate 4 randomly selected minicolumns by setting their output to zero. Then, we similarly report the IRR and the FRR for the real GPU.

We can draw two observations from these experiments. The first one is the graceful degradation of the model to defects. The IRR experiment emulates either transient faults or permanent defects where no retraining was performed. After retraining, the recognition rate remains almost perfect until the number of faulty shader cores per multiprocessor equals 5; then 20 out of 32 minicolumn threads have been disabled, and the remaining 12 functional minicolumns are not numerous enough for the 15 features to be learned. The second observation is that the GPGPUSim simulator with faulty shaders behaves almost exactly the same as the real GPU with deactivated minicolumns. Hence, for subsequent large-scale experiments, we will emulate faulty shaders on a real GPU with deactivated minicolumns.

4.5 Spatially Distributed Defects

In this section, we study the impact of faulty shaders which are randomly spatially distributed. We construct a 6-level hierarchical network. This network contains 24 hypercolumns at level 0, 12 hypercolumns at level 1, 6 hypercolumns at level 2, 3 hypercolumns at level 3, 1 hypercolumn at level 4 and 1 hypercolumn at level 5 for a total of 47 hypercolumns and 940 minicolumns (each hypercolumn contains 20 minicolumns). This network is trained again on a sample of handwritten digits (0-9) obtained from the MNIST database until it achieves 100% recognition rate. At this point, we deactivate shader cores from random locations. Since there are 20 minicolumns within a hypercolumn, deactivating one shader core affects 2.5 minicolumns per hypercolumn on average. Since on average 3.3 out of 47 hypercolumns map onto a single GeForce 9800 GT multiprocessor, each deactivated shader core affects 8.5 minicolumns. For example, for the GeForce 9800 GT, deactivating 6.25% of the shader cores means 7 out of 112 shader cores are deactivated. Thus, a total of around 60 minicolumns throughout the hierarchy are disabled. After deactivating the shader cores,

we evaluate IRR. We then retrain the network until it achieves a stable recognition rate and measure the FRR. The results for this experiment are presented in Figure 8. We can observe the same behavior as in the validation experiment, though it can be noted that 100% recognition accuracy can be achieved even with only 50% functional shader cores.

We also want to illustrate that the robustness of the model can directly benefit from redundant parallel hierarchies without any special algorithmic modification. We create 3 parallel hierarchical networks similar to the one described above. Each hypercolumn within these hierarchies contains 20 minicolumns. The output of each of these hierarchies is fed to an association network which pools minicolumns in the top level of each of the hierarchies firing for the same digit. Thus, if a minicolumn corresponding to a digit in any of the 3 hierarchies fires, the minicolumn in the association network associated to the digit will fire. Essentially, a minicolumn in the association network can be thought of as an 'OR' of of its inputs from the each of the hierarchical networks.

Each of these 3 hierarchies is trained on a sample of MNIST digits. After training, we repeat the same shader core deactivation process as described above. For this case, deactivating a single shader core once again affects 2.5 minicolumns per hypercolumn. But for this case 10 hypercolumns map onto a single multiprocessor. Thus deactivating a single shader core affects approximately 25 minicolumns throughout the 3 hierarchical networks. Figure 9 shows the IRR and FRR for this experiment. Comparing with Figure 8, we see that the IRR of the larger hierarchical network degrades much more slowly and its FRR is similarly improved. From this result, we can also infer that similar benefits can be achieved by simply increasing number of minicolumns per hypercolumn.

4.6 Spatially Clustered Defects

Because hardware defects can also occur in spatial clusters, we evaluate the impact of such defects on the robustness of the model. In the model, neighbor minicolumns are more likely to carry similar information, or to interact in an inhibitory fashion. As a result, clustered defects can potentially be more harmful to the task functionality. On the other hand, the fact that the information quickly spreads out across hierarchy levels can compensate for that vulnerability. We assess the sensitivity to clustered defects in the following experiment. We deactivate 5 neighbor shader cores at a time and plot the results in Figure 10. We can see that if the deactivated shader cores are used for minicolumns higher in the hierarchy, it takes fewer retraining iterations than if they the minicolumns are lower in the hierarchy. This is mainly due to the fact that if a minicolumn at level n is disabled, all the hypercolumns above level n getting input from the disabled minicolumn have to retrain themselves to relearn the lost feature. This is an incremental process: first, the hypercolumn in level n will relearn the lost feature; then, given the new activation pattern in level n minicolumns, level $n+1$ hypercolumns must relearn this pattern; this learning is repeated all the way to the top of the hierarchy.

4.7 Other GPU Defects

Apart from shader core defects, a number of other GPU components e.g. datapath, storage, and scheduler can incur defects.

Datapath Defects: A faulty datapath means that all the shader cores within the affected multiprocessor get corrupt input data. As a result, all the corresponding shaders must be deactivated, and the defect is equivalent to clustered shader defects, investigated in Section 4.6.

Storage Defects: Faulty storage can occur either within shader cores (registers), or at the level of caches or memory banks. In both cases, if ECC cannot compensate for the permanent defects, one or several shader cores must be deactivated. In the former case, one shader must be deactivated. In the latter case, the set of shader cores which can fetch data from/store data into the corresponding caches and banks can be deactivated. Within a multiprocessor, there is a shared memory accessed by all the shader cores, and each of the shader cores has its local memory as well. Permanent local memory defects can be dealt with by deactivating individual shader cores. Shared memory defects can result in the deactivation of the whole multiprocessor, but not the whole GPU. Apart from local and shared memories, there are also global, constant, and texture memories. All the shader cores can access any of these memories. Thus, non-correctable defects in these memories can result in the loss of the entire GPU.

Thread Scheduler Defects: A faulty thread scheduler can manifest itself in a number of ways: sending threads to the wrong shader cores or multiprocessors, sending wrong memory addresses to the shader cores, etc. These cases can be again handled by identifying and deactivating the effected shader cores. Another solution is to consider the scheduler's relative footprint as small enough to afford a robust implementation using larger devices.

Overall, almost all GPU defects can be dealt with by deactivating one or multiple (distributed or clustered) shader cores, even if they do not directly affect shader cores themselves.

5. CONCLUSIONS AND FUTURE WORK

In this article, we advocate leveraging new advances in neuroscience to investigate computing systems which process information by building and then manipulating increasingly abstract representations of information, rather than quickly performing a large set of elementary computations. While the application scope of these computing systems is restricted compared to traditional computers, we know the biological example can significantly outperform traditional computers for a range of tasks; moreover, this computing approach blends well with upcoming limitations of technology (speed, power dissipation, reliability). In the short to medium term, commodity GPGPUs are a promising substrate for deploying large-scale cortical models based on these principals, while direct silicon implementations are still unavailable. We demonstrate that the inherent fault tolerance of cortical networks maps cleanly to failure modes that can occur in commodity GPGPUs, and develop a stuck-at fault model for such systems. Fault injection experiments validate our intuition that such systems are inherently far more tolerant to permanent faults than conventional computing systems.

In the future, we plan to explore a number of directions. First, we want to add other cortical features like feedback, attention, temporal learning, and memory to our learning model. Second, we plan to study the performance of our model on other interesting workloads like game-bots, speech to text conversion, etc. We also plan to study the mapping of our cortical model to hardware circuit implementation.

6. ACKNOWLEDGMENTS

We wish to thank Andrew Nere for many fruitful discussions on the GPGPUSim model, as well as the paper's anonymous reviewers for their helpful comments. This work was supported in part by National Science Foundation award CCF-0702272, as well as equipment donations from Hewlett Packard.

7. REFERENCES

[1] R. Ananthanarayanan and D. S. Modha. Anatomy of a cortical simulator. In *Proceedings of Supercomputing 07*, November 2007.

[2] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamondt. Analyzing cuda workloads using a detailed gpu simulator. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 163–174, 2009.

[3] H. Berry and O. Temam. Modeling self-developping biological neural network. *Neurocomputing*, 70(16-18):2723–2734, 2007.

[4] R. E. Brown and P. M. Milner. The legacy of Donald O. Hebb: more than the Hebb synapse. *Nat Rev Neurosci*, 4(12):1013–1019, Dec 2003.

[5] N. Corporation. *CUDA Programming Guide*. NVIDIA Corporation, 2701 San Toman Expressway, Santa Clara, CA 95050, USA, 2007.

[6] Systems of neuromorphic adaptive plastic scalable electronics (synapse). http://www.darpa.mil/dso/solicitations/baa08-28.html.

[7] A. Dehon. Nanowire-based programmable architectures. *J. Emerg. Technol. Comput. Syst.*, 1(2):109–162, 2005.

[8] M. Emmerson and R. Damper. Determining and improving the fault tolerance of of multilayer perceptrons in a pattern-recognition application, 1993.

[9] J. Fieres, K. Meier, and J. Schemmel. A convolutional neural network tolerant of synaptic faults for low-power analog hardware. In *ANNPR*, pages 122–132, 2006.

[10] S. Haeusler and W. Maass. A statistical analysis of information-processing properties of lamina-specific cortical microcircuit models. *Cereb. Cortex*, 17(1):149–162, Jan 2007.

[11] A. Hashmi, H. Berry, O. Temam, and M. H. Lipasti. Leveraging progress in neurobiology for computing systems. In *Proceedings of the Workshop on New Directions in Computer Architecture held in Conjunction with 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*, 2009.

[12] A. Hashmi and M. Lipasti. Cortical columns: Building blocks for intelligent systems. In *Proceedings of the Symposium Series on Computational Intelligence*, pages 21–28, 2009.

[13] A. Hashmi and M. Lipasti. Discovering cortical algorithms. In *Proceedings of the International Conference on Neural Comp utation (ICNC)*, October 2010.

[14] S. Haykin. *Neural Networks: A Comprehensive Foundation (2nd Edition)*. Prentice Hall, 1998.

[15] J. Hirsch and L. Martinez. Laminar processing in the visual cortical column. *Current Opinion in Neurobiology*, 16:377–384, 2006.

[16] M. Holler, S. Tam, H. Castro, and R. Benson. An electrically trainable artificial neural network (ETANN) with 10240 'floating gate' synapses. In *International Joint Conference on Neural Networks, IJCNN*, volume 2, pages 191–196, Jun 1989.

[17] D. Hubel and T. Wiesel. Receptive fields, binocular interactions and functional architecture in cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962.

[18] K. Ibata, Q. Sun, and G. Turrigiano. Rapid Synaptic Scaling Induced by Changes in Postsynaptic Firing. *Neuron*, 57(6):819–826, 2008.

[19] M. H. Kalisman N, Silberberg G. The neocortical microcircuit as a tabula rasa. *Proc. Natl. Acad. Sci. USA*, 102, 880-885, 2005.

[20] E. Kandel, J. Schwartz, and T. Jessell. *Principles of Neural Science*. McGraw-Hill, 4 edition, 2000.

[21] G. Kreiman, C. Koch, and I. Fried. Category-specific visual responses of single neurons in the human medial temporal lobe. *Nature Neuroscience*, 3:946–953, 2000.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[23] A. Losonczy and J. Magee. Integrative properties of radial oblique dendrites in hippocampal ca1 pyramidal neurons. *Neuron*, 50:291–307, 2006.

[24] A. Nere and M. Lipasti. Cortical architectures on a gpgpu. In *GPGPU '10: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 12–18, New York, NY, USA, 2010. ACM.

[25] J. T. Paz, S. Mahon, P. Tiret, S. Genet, B. Delord, and S. Charpier. Multiple forms of activity-dependent intrinsic plasticity in layer V cortical neurones in vivo. *The Journal of Physiology*, 587(13):3189–3205, 2009.

[26] J. Peissig and M. Tarr. Visual object recognition: do we know more now than we did 20 years ago? *Annu. Rev. Psychol.*, 58:75–96, 2007.

[27] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2:1019–1025, 1999.

[28] D. Ringach. Haphazard wiring of simple receptive fields and orientation columns in visual cortex. *J. Neurophysiol.*, 92(1):468–476, Jul 2004.

[29] S. Ryoo, C. Rodrigues, S. Babhsorkhi, S. Stone, D. Kirk, and W. Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *Proceesings Symposium on principles and practices of parallel programming, SIGPLAN*, pages 73 –82, 2008.

[30] A. Sandberg and N. Bostrom. Whole brain emulation: A roadmap. Technical Report 2008-3, Future of Humanity Institute, Oxford University, 2008.

[31] J. Schemmel, J. Fieres, and K. Meier. Wafer-scale integration of analog neural networks. In *IEEE International Joint Conference on Neural Networks, IJCNN*, pages 431–438, June 2008.

[32] T. Serre, A. Oliva, and T. Poggio. A feedforward architecture accounts for rapid categorization. *Proc. Natl. Acad. Sci. USA*, 104(15):6424–6429, Apr 2007.

[33] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):411–426, Mar 2007.

[34] SIA. Semiconductor industry association 2007 roadmap. http://www.sia-online.org/, 2007.

[35] G. Sperling. A model for visual memory tasks. *Human Factor*, 5:19–31, 1963.

[36] M. Spratling. Presynaptic lateral inhibition provides a better architecture for self-organising neural networks. *Network: Computation in Neural Systems,*, 10(4):285–301, 1999.

[37] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[38] L. Swanson. Mapping the human brain: past, present, and future. *Trends in Neurosciences*, 18(11):471 –474, 1995.